Final Project Part 2

Medical Practice Database

Alex Bordanca, David Thiriot

Create Conceptual Diagram / Schema for database

The database we chose to develop is intended to store data about patients and doctors as might be used in a medical practice. After searching for different datasets, we chose to simulate (invent) data on 1000 patients and 10 doctors. The data includes personal information, address and contact information, health information, and financial information. The information included in the database includes all the following (35 information fields):

Patient ID

Whether the patient is a current patient \* (in contrast to a former patient)

Patient First Name

Patient Last Name

Patient Date of Birth

Patient Age

Patient Biological Sex (Male / Female)

Patient Ethnicity

Patient most recent height

Patient most recent weight

Patient most recent heart rate

Patient most recent systolic blood pressure

Patient most recent diastolic blood pressure

Patient Email address

Patient Street address

Patient City

Patient State

Patient Zip code

Patient Account balance due

Patient Insurance company

Whether the patient currently smokes

Whether the patient has each of 10 health conditions called “Condition 1” through “Condition 10”. (The incidence rate of these conditions ranges from 30% of the patient population for Condition 1, to 1% of the patient population for Condition 10.)

Doctor that each patient is assigned to

Doctor ID number for each of 10 doctors

Doctor First name

Doctor Last name

\* The Current\_patient field was added as a new column in a table after database creation, to demonstrate the ALTER TABLE and UPDATE functions of SQLite.

While our initial intent was to create a list of very generic names and addresses (Patient1, Patient2, etc.), we found a way using R to simulate (invent) realistic names, addresses, email addresses, and health information. Our R code for simulating the data is included in an appendix to this report.

To normalize the data, we separated the data into 7 tables. The name of each table, the columns in each table and the SQLite constraints on each column are shown below.

TABLE Doctors

Doctor\_ID int PRIMARY KEY

Firstname text

Lastname text

TABLE Patient\_contact

Patient\_ID int PRIMARY KEY

Email text

Street\_address text

City text

State text

Zip int

TABLE Patient\_doctors

Patient\_ID int PRIMARY KEY

Doctor text NOT NULL (This is the doctor’s last name)

TABLE Patient\_health

Patient\_ID int PRIMARY KEY

Current\_smoker int (In many cases, we are using int as a boolean 0/1 = False/True.)

Condition\_1 int

Condition\_2 int

Condition\_3 int

Condition\_4 int

Condition\_5 int

Condition\_6 int

Condition\_7 int

Condition\_8 int

Condition\_9 int

Condition\_10 int

TABLE Patient\_ID

Patient\_ID int PRIMARY KEY

Firstname text NOT NULL (At least 1 name is required for a patient.)

Lastname text

DOB text

Age int

Biol\_sex text

Ethnicity text (The simulated names match with the sex and ethnicity.)

Current\_patient int (\*Added after initial database creation using ALTER TABLE.)

TABLE Patient\_vitals

Patient\_ID int PRIMARY KEY

Last\_height real

Last\_weight real

Last\_heartrate real

Last\_systolic\_BP real

Last\_diastolic\_BP real

Database

The SQLite database that we create is called “MedPracDB.db”, and is comprised of 7 tables. Each patient is assigned a unique Patient ID number, and that Patient ID number serves as the Primary Key for 6 of the 7 tables. Doctors at the practice are also assigned a unique Doctor ID number, which serves as a Primary Key for the Doctors table.

Additional constraints on the database include the need for each patient to have a doctor at the practice (NOT NULL) and that each patient must have at least one name (First name NOT NULL).

After we realized that it is realistic to expect that a medical practice would retain medical records for a period of time, even after a person is no longer a patient at the practice, we used ALTER TABLE to add the column Current\_patient, an integer acting as a boolean flag as to whether the patient is current (1) or former(0). Patients can thus be “deleted” from visibility in queries, without being fully removed from the database.

To demonstrate that the PRIMARY KEY constraints are working as intended, in our code we attempt to enter some new patients into the database using existing Patient\_ID numbers. As should be the case, this is not allowed, and the message “IntegrityError: column Patient\_ID is not unique” is returned, maintaining the integrity of our database.

We learned that by default, SQLite does not recognize Foreign Keys. We chose not to enable that functionality, and to run the database without explicitly indicating Foreign Keys for tables.

Code Demonstrating Database Functions

See code file MedPracDB.ipynb submitted together with this Word document.

In summary, the Python code does the following:

* Create the database
* Create 7 tables, specifying constraints, and read the appropriate simulated data into each table
* Handle error such as a table or column already existing before a CREATE attempt
* Add a new column to one of the tables, and set values of rows in the column
* Create a function that adds new patients to the database from rows in a Pandas dataframe.
* Add new patients to the database (adds rows to 6 different tables in a function)
* Show that records with non-unique Patient\_ID numbers can’t be added
* Demonstrate interesting queries using multiple JOINS such as
  + Identify all the female patients of one of the doctors, who have medical “Condition 1” and a certain insurance company. Return their email, so they can be contacted about a change in coverage by their insurance company for that condition.
  + Identify all the patients at the practice who have rare medical “Condition 10”.
  + Determine correlation between whether a patient is a smoker and the amount they owe the practice
* Change a current patient to a former patient, removing them from visibility in certain queries
* Create a density plot of the height of all patients, separated by Male and Female
* Show the height of a particular patient, in the context of all patients at the practice
* Create a height/weight distribution plot by gender
* Create a list of correlations between other vitals and a dummy BMI variable

Individual Assessment / Evaluation of Work

The use of a simulated dataset, appealed to me and I was quickly onboard with the idea which was initially presented by David. It eased the normalization process allowing us to start development of our actual web app. With that being said, though, the normalization of the database increased the complexity of the design of the web app. As we are using Django, wherein tables translate to data models, and views are functions that render web pages for different features of interacting with the data, there were many more views and html pages to create than if there would have been only one table.

Using SQLite seemed intuitive, as ultimately we would have needed to migrate any DBMS we would have used to SQLite for the Heroku hosting, so this naturally skips that step. The one limitation that I’ve been running into in the brunt of the Django work is dealing with the lack of a foreign key constraint, which makes data retrieval, modification, and deletion a little more complex. We have devised somewhat of a solution for these limitations, but personally I am not fully pleased with them and will continue to work to improve it

David Thiriot personal assessment

The decision to simulate a dataset came after looking at several datasets that did not seem ideal for this project. Some datasets we considered would not have generated enough tables (they were too simple); others were much too large. The combination of not having an ideal dataset, and interest in the idea to simulate a medical practice, led me to simulate data for 1000 patients. I chose R, because for projects of that scale, it is still seems a little quicker and easier for me than Python. I was actually surprised (and scared) that it was so easy to generate thousands of very realistic sounding names, that appropriately matched the sex and ethnicity of the invented persons. I am pleased with that part of the work.

I like using SQLite because it does not require setting up a separate software program and local server connection. Server connections and getting other programs to work with Python has been one of the most challenging parts of this databases course for me.

I attempted to generate a graphical schema from our SQLite database using the package ERAlchemy, but could not get it to properly install on my computer. The same information is captured in tabular form (just a little less visually appealing).

I still have much to learn about effective ways to use the built in constraints available in SQL. For example, as SQLite by default does not recognize foreign keys, we are operating the database without explicit foreign keys. I wonder what additional value could be gained by learning and implementing foreign key restraints more effectively.

Appendix: R code written to simulate the dataset for our Medical Practice database

# SimMedPracticeData.R

# Simulate all the people in a medical practice, and realistic information about them

# To use in a database project

# 15 March 2023

# David Thiriot

rm(list = ls())

cat("\014")

opar <- par()

setwd(PATH REDACTED)

randomseed = 2023

NumberOfPatients <- 1000

MedicalConditions <- c("Condition\_1", "Condition\_2", "Condition\_3", "Condition\_4", "Condition\_5", "Condition\_6", "Condition\_7", "Condition\_8", "Condition\_9", "Condition\_10")

ConditionPrevelance <- c(0.3, 0.25, 0.2, 0.15, 0.1, 0.08, 0.06, 0.04, 0.02, 0.01)

PatientColumns <- c("Patient\_ID", "Lastname", "Firstname", "Biol\_sex", "Ethnicity", "DOB", "Age", "Doctor", "Street\_address", "City", "State", "Zip", "Email", "Ins\_co", "Amount\_due", "Last\_height", "Last\_weight", "Last\_heartrate", "Last\_systolic\_BP", "Last\_diastolic\_BP", "Current\_smoker", MedicalConditions)

patient\_df <- data.frame(seq(from=1, to=NumberOfPatients))

colnames(patient\_df) <- "Patient\_ID"

##### Add Name, Biological sex, ethnicity using R package randomNames

#install.packages("randomNames")

library(randomNames)

set.seed(randomseed)

fakenames <- randomNames(n=NumberOfPatients, which.names = "both", name.order = "last.first", return.complete.data = TRUE)

patient\_df$Lastname <- fakenames$last\_name

patient\_df$Firstname <- fakenames$first\_name

patient\_df$Biol\_sex = "Male"

patient\_df$Biol\_sex[fakenames$gender=="1"] <- "Female"

patient\_df$Ethnicity <- "American Indian or Native Alaskan"

patient\_df$Ethnicity[fakenames$ethnicity==2] <- "Asian or Pacific Islander"

patient\_df$Ethnicity[fakenames$ethnicity==3] <- "Black (not Hispanic)"

patient\_df$Ethnicity[fakenames$ethnicity==4] <- "Hispanic"

patient\_df$Ethnicity[fakenames$ethnicity==5] <- "White (not Hispanic)"

patient\_df$Ethnicity[fakenames$ethnicity==6] <- "Middle-Eastern, Arabic"

##### Add random date of birth within a range

set.seed(randomseed)

# reference for code idea = Joshua Ulich, 07Jan2013, accessed 17March2023 at

# https://stackoverflow.com/questions/14201530/what-is-a-good-way-to-select-random-dates-over-a-given-interval-using-r

Start <- as.Date("1943-01-01")

End <- as.Date("2003-01-01")

DsOB <- Start + sample.int(End-Start, NumberOfPatients)

patient\_df$DOB = DsOB

#install.packages("lubridate")

library(lubridate)

ages = year(Sys.Date()) - year(DsOB)

patient\_df$Age <- ages

#####

# Assign a Doctor

set.seed(randomseed)

# I am making up 10 random doctor names

doctors\_lastname = c("Smith", "Williams", "Jones", "Rodriguez", "Zhang", "Perez", "Jackson", "Harris", "Tataryn", "Petit")

doctors\_firstname = c("Michael", "Sally", "Dennis", "Juana", "Li", "Ernesto", "Veronica", "Patricia", "Suleman", "Alain")

assigned\_doc <- sample(doctors\_lastname, NumberOfPatients, replace=TRUE)

patient\_df$Doctor = assigned\_doc

##### Create a separate .csv for Doctor information

###

doctors\_df <- data.frame(seq(from=1, to=10))

colnames(doctors\_df) <- "Doctor\_ID"

doctors\_df$Firstname = doctors\_firstname

doctors\_df$Lastname = doctors\_lastname

write.csv(doctors\_df, "Doctor data.csv", row.names=FALSE)

#####

# Add random mailing addresses and email address

#install.packages("charlatan")

library(charlatan)

set.seed(randomseed)

z <- AddressProvider$new('en\_US')

for(counter in 1:NumberOfPatients){

patient\_df$Street\_address[counter] = z$street\_address()

patient\_df$City[counter] = "New York"

patient\_df$State[counter] = "New York"

patient\_df$Zip[counter] = sample(c(10013, 10007, 10006, 10005, 10002, 10012), 1, replace=TRUE)

}

#####

# Add email address

set.seed(randomseed)

for(counter in 1:NumberOfPatients){

patient\_df$Email[counter] = paste0(substr(patient\_df$Lastname[counter],1,3),substr(patient\_df$Firstname[counter],1,3), sample((1:9), 1, replace=TRUE), "@", sample(c("gmail.com", "verizon.net", "yahoo.com", "gmail.com", "aol.com", "fastmail.net"), 1, replace=TRUE))

}

#####

# Add Insurance Company and placeholder for account balance (will fill in balance later based on medical conditions)

set.seed(randomseed)

patient\_df$Ins\_co = sample(c("LifeWell", "Healthplan Plus", "New Day Medical", "Health Partners of New York", "CityPlan Health", "MetroCare Gold", "MetroCare Basic"), NumberOfPatients, replace=TRUE)

patient\_df$Amount\_due = 0

#####

# Add height (cm), weight (lbs), heartrate (bpm), systolic\_BP, diastolic\_BP, based on realistic ranges for male and female

set.seed(randomseed)

for(counter in 1:NumberOfPatients){

if(patient\_df$Biol\_sex[counter]=="Male"){

patient\_df$Last\_height[counter]=round(rnorm(1, mean=175.26, sd=7.62), 0)

patient\_df$Last\_weight[counter]=round(rnorm(1, mean=210, sd=10),0)

patient\_df$Last\_heartrate[counter]=round(rnorm(1, mean=71, sd=7),0)

patient\_df$Last\_systolic\_BP[counter]=round(rnorm(1, mean=124, sd=5),0)

patient\_df$Last\_diastolic\_BP[counter]=round(rnorm(1, mean=72, sd=5),0)

patient\_df$Current\_smoker[counter]=rbinom(1,1,0.167)

}

else{

patient\_df$Last\_height[counter]=round(rnorm(1, mean=162.56, sd=7),0)

patient\_df$Last\_weight[counter]=round(rnorm(1, mean=165, sd=10),0)

patient\_df$Last\_heartrate[counter]=round(rnorm(1, mean=80, sd=7),0)

patient\_df$Last\_systolic\_BP[counter]=round(rnorm(1, mean=121, sd=5),0)

patient\_df$Last\_diastolic\_BP[counter]=round(rnorm(1, mean=70, sd=5),0)

patient\_df$Current\_smoker[counter]=rbinom(1,1,0.136)

}

}

#####

# Simulate whether each patient has any of 10 adverse health conditions, based on assumed values for prevelance

set.seed(randomseed)

patient\_df$Condition\_1 = rbinom(NumberOfPatients, 1, ConditionPrevelance[1])

patient\_df$Condition\_2 = rbinom(NumberOfPatients, 1, ConditionPrevelance[2])

patient\_df$Condition\_3 = rbinom(NumberOfPatients, 1, ConditionPrevelance[3])

patient\_df$Condition\_4 = rbinom(NumberOfPatients, 1, ConditionPrevelance[4])

patient\_df$Condition\_5 = rbinom(NumberOfPatients, 1, ConditionPrevelance[5])

patient\_df$Condition\_6 = rbinom(NumberOfPatients, 1, ConditionPrevelance[6])

patient\_df$Condition\_7 = rbinom(NumberOfPatients, 1, ConditionPrevelance[7])

patient\_df$Condition\_8 = rbinom(NumberOfPatients, 1, ConditionPrevelance[8])

patient\_df$Condition\_9 = rbinom(NumberOfPatients, 1, ConditionPrevelance[9])

patient\_df$Condition\_10 = rbinom(NumberOfPatients, 1,ConditionPrevelance[10])

#####

# Calculate Amount\_due based on medical conditions

ChargePerCondition = c(100,200,300,400,500,1000,2000,3000,5000,10000)

for(counter in 1:NumberOfPatients){

patient\_df$Amount\_due[counter] = sum(ChargePerCondition \* as.numeric(patient\_df[counter, 21:30]))

}

write.csv(patient\_df, file="Simulated Medical Practice Data.csv", row.names=FALSE)

#####

# Create Tables normalized that each focus on certain data

doctors\_table <- doctors\_df

write.csv(doctors\_table, "doctors table.csv", row.names=FALSE)

patient\_ID\_table <- patient\_df[c("Patient\_ID", "Firstname", "Lastname", "DOB", "Age", "Biol\_sex", "Ethnicity")]

write.csv(patient\_ID\_table, "patient ID table.csv", row.names=FALSE)

patient\_doctor\_table <- patient\_df[c("Patient\_ID", "Doctor")]

write.csv(patient\_doctor\_table, "patient doctors table.csv", row.names=FALSE)

patient\_finance\_table <- patient\_df[c("Patient\_ID", "Amount\_due", "Ins\_co")]

write.csv(patient\_finance\_table, "patient finance table.csv", row.names=FALSE)

patient\_contact\_table <- patient\_df[c("Patient\_ID", "Email", "Street\_address", "City", "State", "Zip")]

write.csv(patient\_contact\_table, "patient contact table.csv", row.names=FALSE)

patient\_vitals\_table <- patient\_df[c("Patient\_ID", "Last\_height", "Last\_weight", "Last\_heartrate", "Last\_systolic\_BP", "Last\_diastolic\_BP")]

write.csv(patient\_vitals\_table, "patient vitals table.csv", row.names=FALSE)

patient\_health\_table <- patient\_df[c("Patient\_ID", "Current\_smoker", "Condition\_1", "Condition\_2", "Condition\_3", "Condition\_4", "Condition\_5", "Condition\_6", "Condition\_7", "Condition\_8", "Condition\_9", "Condition\_10")]

write.csv(patient\_health\_table, "patient health table.csv", row.names=FALSE)